REMARKS

No claims are amended. No new claims are added. Claims 1-45 are pending for consideration. In view of the following remarks, Applicant respectfully requests that this application be allowed and forwarded on to issuance.

The § 102 Rejection

Claim 39 stands rejected under 35 U.S.C. § 102(e) as being anticipated by U.S. Pub. No. 2002/0026461 A1 (hereinafter "Kutay").

The § 103 Rejections

Claims 1, 3-9 and 20-26 stand rejected under § 103(a) as being unpatentable by Kutay in view of the Internet Web Sites http://www.w3.org/TR/1999/, editor James Clark "W3C XSL Transformations (XSLT) Version 1.0" (hereinafter "Clark").

Claim 2 stands rejected under § 103(a) as being unpatentable over Kutay in view of Clark and U.S. Pub No 2003/0120659 A1 (hereinafter Sridhar).

Claims 10-19 stand rejected under § 103(a) as being unpatentable by Clark in view of Kutay.

Claims 27-29, 32-38, 40, 41 and 43 stand rejected under § 103(a) as being unpatentable by Kutay in view of the Internet Web Sites http://www.w3.org/TR/1999/, Editor James Clark and Steve Derose "W3C XML Path Language (Xpath) Version 1.0" (hereinafter "Derose").

Claims 30, 31, 42, 44 and 45 stand rejected under § 103(a) as being unpatentable over Kutay in view of Derose and Clark.

Applicant's Disclosure

Applicant's disclosure is directed to methods and systems of authoring XML using DHTML views. When a user interacts with a DHTML document for the purpose of changing, in some way, the document through either manipulation of one or more of its values or properties, it is important that those manipulations be made, in a consistent manner, to the XML document that describes the structure of the data behind the DHTML document. In order to manipulate the XML document that describes the structure of the data behind the DHTML, there needs to be a way to transform user interactions in the DHTML to changes in the XML document. This is the problem of finding an inverse of the transformation function that is provided by the XSL-T.

In one implementation, the disclosure addresses this problem by automatically (or semi-automatically, with some hint given by the application developer) generating an appropriate user interface (UI) within the DHTML document that allows the user to manipulate or interact with the DHTML document. The presentation of the UI takes into account not only the XML schema, ***but also the XSL-T transformations that were utilized to provide the DHTML***. This represents a significant departure from other XML authoring solutions that look only to the XML schema to determine what can and cannot be added to an XML document. The UIs thus allow user interaction with the DHTML view (e.g. adding and/or deleting ***structure***) to be directly transferred back to the XML document.

In various embodiments, a decision process is undertaken that decides which UIs to present to a user and when to present them. That is, there are potentially a number of different UI choices that can be made depending on what a user is doing in a particular document and where they are in the document. An inventive approach is to utilize a number of different parameters and based upon

analysis of the parameters make a decision on which UI to present to a user so that they can interact with the DHTML view. In various embodiments, the following parameters can be used:

- Selection of where a user is in a particular DHTML document. This translates to where a user is in a particular XML document because the selection initially starts on the DHTML side and has a correspondence on the XML side;
- The portion of the XML schema that corresponds to the user's selection;
- The UI types that would be desirable to generate; and
- ***The XSL-T file***

In the XSL-T file, there are certain constructs that can be suggestive of certain structures in the resultant DHTML. For example, the XSL-T file may include a "xsl:for-each" construct (i.e. for each customer, take a certain action). This construct is suggestive of a repetitive structure in the DHTML, such as a table or a paragraph. That is, if there are a number of customers, then repeating a certain action would repetitively define a certain type of structure. By considering these XSL-T constructs, certain UI types can be identified that can be displayed for the user.

An example is table editing. For example, if expenses are optional, according to the schema, initially there may be no expenses in a table. The XSL-T would have a "for-each" construct to render each expense, but since there are none in the XML doc, nothing is displayed. The UI should in this case produce a context block for adding an expense.

Once the first expense is created, by re-applying the XSLT transformation, a table is now viewable. At this point, based on the XSL-T hint that there is a "for-each" associated with an expense, and the schema information that multiple expenses can be added, a decision is made to not show the "Add expense" as a

context block, but to add an appropriate in-doc UI that would now take over the functionality of adding additional expenses as new rows to the table.

In order to provide these types of UIs, embodiments examine the XSL-T file to identify which UI candidates are more suited to have their functionalitics provided by "in document" UIs. For example, if the schema specifies that multiple expenses are allowed, and the XSL-T has a "for-each" construct for expenses, by looking at the first element introduced by the XSL-T after an expense is matched, it could determine what kind of helpful UI to add. If an DHTML TABLE is created, then it should be adorned with table-editing widgets, but if there is SPAN, for example, then create a context block, and not an in-doc UI.

Claims 1-9

Claim 1 recites a method of providing a user interface (UI) comprising [emphasis added]:

- rendering a DHTML document from an XML document using at least one XSLT transformation (XSL-T); and
- presenting a user interface ***based, at least in part, on the XSL-T*** that was used to render the DHTML document.

In making out the rejection of claim 1, the Office argues that Kutay teaches "user interface module 222 to create views 432 presented in XML format and an XML transform editor 436 is further provided to convert documents created in a source format from a source Document Type Definition (DTD), for example XML, to a target DTD, for example HTML, and to present the document to users in the target format defined by the target DTD." The Office further argues that Kutay teaches "HTML and Java server-side technology to create dynamic, interactive pages." The Office cites to paragraphs 62 and 85 in support of its arguments, both of which are reproduced below:

> [0062] In one embodiment, user interface module 222 further includes a view editor 433 to create one or more views 432 within the presentation layer 430 of the application 400 and an action editor 435 to define actions 434 within the presentation layer 430. In one embodiment, an XML editor 437 is provided within user interface module 222 to create views 432 presented in XML format and an XML transform editor 436 is further provided to convert documents created in a source format from a source Document Type Definition (DTD), for example XML, to a target DTD, for example HTML, and to present the document to users in the target format defined by the target DTD.
>
> [0085] As illustrated in FIG. 6C, in one embodiment, the views displayed to the user 205 are JSP views, which use HTML and Java server-side technology to create dynamic, interactive pages. In one embodiment, user 205 submits a request across network 100 using input view 601 containing input parameters. Application server 210 receives the request and passes details of the action 611 specified within input view 601 to client 220.

The Office asks Applicant to compare the cited excerpts with Applicant's claimed features, namely, "rendering a DHTML document from an XML document; and presenting a user interface based, at least in part, on the XSL-T that was used to render the DHTML document."

Applicant has done as the Office requests and respectfully submits that Kutay does not teach or suggest presenting a user interface ***based, at least in part, on the XSL-T*** that was used to render the DHTML document. In fact, the Office ***admits*** that Kutay does not explicitly teach rendering a document using at least one XSL-T. It logically follows that if Kutay does not render a DHTML document using at least one XSLT transformation, it cannot possibly present a user interface ***based, at least in part, on the XSL-T*** that was used to render the DHTML document. Furthermore, Clark does not disclose or suggest the missing feature. Accordingly, for at least this reason, the Office has failed to establish a *prima facie* case of obviousness and this claim is allowable.

Claims 2-9 depend from claim 1 and, as such, are allowable as depending from an allowable base claim. These claims are also allowable for their own recited features which, in combination with those recited in claim 1, are neither shown nor suggested by the references of record, either singly or in combination with one another. In addition, given the Office's failure to establish a *prima facie* case of obviousness, the rejection of claim 2 over Sridhar is not seen to add anything of significance.

Claims 10-19

Claim 10 recites a method of providing a user interface comprising [emphasis added]:

- considering multiple parameters ***one of which includes an XSL-T file***; and
- ***based upon the considered parameters***, rendering a user interface sufficient to enable a user to interact with a DHTML view that has been rendered by the XSL-T file from an XML document.

In making out the rejection of this claim, the Office argues that Clark teaches "parameters are passed to templates using the XSL." The Office further argues that Clark teaches "the result tree will contain a character that cannot be represented in the encoding that the XSLT processor is using for output." The Office cites to sections 2.2, 11.6, and 16.2 in support of its arguments, all of which are reproduced below:

2.2 Stylesheet Element

```
<xsl:stylesheet
  id = id
  extension-element-prefixes = tokens
  exclude-result-prefixes = tokens
  version = number>
  <!-- Content: (xsl:import*, top-level-elements) -->
</xsl:stylesheet>
```

```
<xsl:transform
  id = id
  extension-element-prefixes = tokens
  exclude-result-prefixes = tokens
  version = number>
  <!-- Content: (xsl:import*, top-level-elements) -->
</xsl:transform>
```

A stylesheet is represented by an `xsl:stylesheet` element in an XML document. `xsl:transform` is allowed as a synonym for `xsl:stylesheet`.

An `xsl:stylesheet` element must have a `version` attribute, indicating the version of XSLT that the stylesheet requires. For this version of XSLT, the value should be `1.0`. When the value is not equal to `1.0`, forwards-compatible processing mode is enabled (see **[2.5 Forwards-Compatible Processing]**).

The `xsl:stylesheet` element may contain the following types of elements:

- `xsl:import`
- `xsl:include`
- `xsl:strip-space`
- `xsl:preserve-space`
- `xsl:output`
- `xsl:key`
- `xsl:decimal-format`
- `xsl:namespace-alias`
- `xsl:attribute-set`
- `xsl:variable`
- `xsl:param`
- `xsl:template`

An element occurring as a child of an `xsl:stylesheet` element is called a **top-level** element.

This example shows the structure of a stylesheet. Ellipses (. . .) indicate where attribute values or content have been omitted. Although this example shows one of each type of allowed element, stylesheets may contain zero or more of each of these el ments.

```
<xsl:stylesheet version="1.0"
```

```
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:import hr f="..."/>

  <xsl:include href="..."/>

  <xsl:strip-space elements="..."/>

  <xsl:preserve-space elements="..."/>

  <xsl:output method="..."/>

  <xsl:key name="..." match="..." use="..."/>

  <xsl:decimal-format name="..."/>

  <xsl:namespace-alias stylesheet-prefix="..."
result-prefix="..."/>

  <xsl:attribute-set name="...">
    ...
  </xsl:attribute-set>

  <xsl:variable name="...">...</xsl:variable>

  <xsl:param name="...">...</xsl:param>

  <xsl:template match="...">
    ...
  </xsl:template>

  <xsl:template name="...">
    ...
  </xsl:template>

</xsl:stylesheet>
```

The order in which the children of the `xsl:stylesheet` element occur is not significant except for `xsl:import` elements and for error recovery. Users are free to order the elements as they prefer, and stylesheet creation tools need not provide control over the order in which the elements occur.

In addition, the `xsl:stylesheet` element may contain any element not from the XSLT namespace, provided that the expanded-name of the element has a non-null namespace URI. The presence of such top-level elements must not change the behavior of XSLT elements and functions defined in this document; for example, it would not be permitted for such a top-level element to specify that `xsl:apply-templates` was to use different rules to resolve conflicts. Thus, an XSLT processor is always free to ignore such top-level elements, and must ignore a top-level element without giving an error if it does not recognize the namespace URI. Such elements can provide, for example,

- information used by extension elements or extension functions (see [**14 Extensions**]),
- information about what to do with the result tree,
- information about how to obtain the source tree,
- metadata about the stylesheet,
- structured documentation for the stylesheet.

11.6 Passing Parameters to Templates

```
<xsl:with-param
  name = qname
  select = expression>
  <!-- Content: template -->
</xsl:with-param>
```

Parameters are passed to templates using the `xsl:with-param` element. The required `name` attribute specifies the name of the parameter (the variable the value of whose binding is to be replaced). The value of the `name` attribute is a QName, which is expanded as described in [**2.4 Qualified Names**]. `xsl:with-param` is allowed within both `xsl:call-template` and `xsl:apply-templates`. The value of the parameter is specified in the same way as for `xsl:variable` and `xsl:param`. The current node and current node list used for computing the value specified by `xsl:with-param` element is the same as that used for the `xsl:apply-templates` or `xsl:call-template` element within which it occurs. It is not an error to pass a parameter *x* to a template that does not have an `xsl:param` element for *x*; the parameter is simply ignored.

This example defines a named template for a `numbered-block` with an argument to control the format of the number.

```
<xsl:template name="numbered-block">
  <xsl:param name="format">1. </xsl:param>
  <fo:block>
    <xsl:number format="{$format}"/>
    <xsl:apply-templates/>
  </fo:block>
</xsl:template>

<xsl:template match="ol//ol/li">
  <xsl:call-template name="numbered-block">
    <xsl:with-param name="format">a. </xsl:with-
param>
  </xsl:call-template>
</xsl:template>
```

16.2 HTML Output Method

The `html` output method outputs the result tree as HTML; for xample,

```
<xsl:stylesheet version="1.0"

xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:output method="html"/>

<xsl:template match="/">
  <html>
   <xsl:apply-templates/>
  </html>
</xsl:template>

...

</xsl:stylesheet>
```

The `version` attribute indicates the version of the HTML. The default value is `4.0`, which specifies that the result should be output as HTML conforming to the HTML 4.0 Recommendation [HTML].

The `html` output method should not output an element differently from the `xml` output method unless the expanded-name of the element has a null namespace URI; an element whose expanded-name has a non-null namespace URI should be output as XML. If the expanded-name of the element has a null namespace URI, but the local part of the expanded-name is not recognized as the name of an HTML element, the element should output in the same way as a non-empty, inline element such as `span`.

The `html` output method should not output an end-tag for empty elements. For HTML 4.0, the empty elements are `area`, `base`, `basefont`, `br`, `col`, `frame`, `hr`, `img`, `input`, `isindex`, `link`, `meta` and `param`. For example, an element written as `<br/>` or `<br></br>` in the stylesheet should be output as `<br>`.

The `html` output method should recognize the names of HTML elements regardless of case. For example, elements named `br`, `BR` or `Br` should all be recognized as the HTML `br` element and output without an end-tag.

The `html` output method should not perform escaping for the content of the `script` and `style` elements. For example, a literal result element written in the stylesheet as

```
<script>if (a &lt; b) foo()</script>
```

or

```
<script><![CDATA[if (a < b) foo()]]></script>
```

should be output as

```
<script>if (a < b) foo()</script>
```

Th `html` output method should not escape < characters occurring in attribut values.

If the `indent` attribute has the value `yes`, then the `html` output method may add or remove whitespace as it outputs the result tree, so long as it does not change how an HTML user agent would render the output. The default value is `yes`.

The `html` output method should escape non-ASCII characters in URI attribute values using the method recommended in

Section B.2.1 of the HTML 4.0 Recommendation.

The `html` output method may output a character using a character entity reference, if one is defined for it in the version of HTML that the output method is using.

The `html` output method should terminate processing instructions with `>` rather than `?>`.

The `html` output method should output boolean attributes (that is attributes with only a single allowed value that is equal to the name of the attribute) in minimized form. For example, a start-tag written in the stylesheet as

```
<OPTION selected="selected">
```

should be output as

```
<OPTION selected>
```

The `html` output method should not escape a `&` character occurring in an attribute value immediately followed by a `{` character (see Section B.7.1 of the HTML 4.0 Recommendation). For example, a start-tag written in the stylesheet as

```
<BODY bgcolor='&amp;{{randomrbg}};'>
```

should be output as

```
<BODY bgcolor='&{randomrbg};'>
```

The `encoding` attribute specifies the preferred encoding to be used. If there is a `HEAD` element, then the `html` output method should add a `META` element immediately after the start-tag of the `HEAD` element specifying the character encoding actually used. For example,

```
<HEAD>
<META http-equiv="Content-Type"
content="text/html; charset=EUC-JP">
...
```

> It is possible that the result tree will contain a character that cannot be repres nted in the encoding that the XSLT processor is using for output. In this case, if the character occurs in a context wh r HTML recognizes character references, then the character should be output as a character entity reference or decimal numeric character reference; otherwise (for example, in a `script` or `style` element or in a comment), the XSLT processor should signal an error.
>
> If the `doctype-public` or `doctype-system` attributes are specified, then the `html` output method should output a document type declaration immediately before the first element. The name following `<!DOCTYPE` should be `HTML` or `html`. If the `doctype-public` attribute is specified, then the output method should output `PUBLIC` followed by the specified public identifier; if the `doctype-system` attribute is also specified, it should also output the specified system identifier following the public identifier. If the `doctype-system` attribute is specified but the `doctype-public` attribute is not specified, then the output method should output `SYSTEM` followed by the specified system identifier.
>
> The `media-type` attributc is applicable for the `html` output method. The default value is `text/html`.

The Office asks Applicant to compare the cited excerpts with Applicant's claimed features, namely, "considering multiple parameters one of which includes an XSL-T file" and "based upon the considered parameters, rendering a user interface sufficient to enable a user to interact with a DHTML view that has been rendered by the XSL-T file from an XML document."

Applicant has done as the Office requests and respectfully submits that Clark does not teach or disclose considering multiple parameters ***one of which includes an XSL-T file*** and then, ***based upon the considered parameters***, rendering a user interface sufficient to enable a user to interact with a DHTML view that has been rendered by the XSL-T file from an XML document. Furthermore, Kutay does not disclose or suggest the missing feature. Accordingly,

the Office has failed to stablish a *prima facie* case of obviousness and, for at least this reason, this claim is allowable.

Claims 11-19 depend from claim 10 and, as such, are allowable as depending from an allowable base claim. These claims are also allowable for their own recited features which, in combination with those recited in claim 10, are neither shown nor suggested by the references of record, either singly or in combination with one another.

Claims 20-26

Claim 20 recites a method of providing a user interface comprising [emphasis added]:

- ***making a selection in a DHTML view;***
- determining, based upon the selection, a ***corresponding selection*** in an XML document;
- determining, based upon the corresponding selection in the XML document, a corresponding portion of an XML schema;
- determining, based upon the XML schema portion, one or more types of action that can be undertaken;
- producing one or more operations that can be undertaken for various determined action types; and
- determining, from an XSL-T file that rendered the DHTML view, a user interface type that can be displayed for a user and used to implement the one or more operations.

In making out the rejection of this claim, the Office argues that Kutay teaches "a view template is created. In one embodiment, user 205 creates an HTML template for a view 432 through view editor 433 within user interface module 222." The Office further argues that Kutay teaches "views 432 may be presented in extensible Markup Language (XML)." The Office cites to paragraphs 58 and 144 in support of its arguments, both of which are reproduced below:

> [0058] Referring back to FIG. 4A, in one embodiment, presentation layer 430 includes multiple views 432 which allow users 205 to view processed data. In one embodiment, views 432 are Java Server Page (USP) views. Each JSP view 432 is a dynamic page, for example an HTML page, which supports event-based input mechanisms and contains special tags interpretable by the server 210. Alternatively, views 432 may be presented in eXtensible Markup Language (XML). In one embodiment, each XML view 432 is an XML document accessible to users 205 via Universal Resource Locators (URLs).
>
> [0144] At processing block 1220, a view template is created. In one embodiment, user 205 creates an HTML template for a view 432 through view editor 433 within user interface module 222.

The Office asks Applicant to compare the cited excerpts with Applicant's claimed features, namely, "making a selection in a DHTML view and determining, based upon the selection, a corresponding selection in an XML document."

Applicant has done as the Office requests and respectfully submits that Kutay does not teach or disclose making a selection in a DHTML view and determining, based upon the selection, a ***corresponding*** selection in an XML document. Rather, Kutay simply creates an HTML template for a view which may be presented in XML. There is nothing in Kutay to even suggest making a selection in the view itself, and then determining, from that selection, a corresponding selection in an XML document. Furthermore, Clark does not disclose or suggest the missing feature. Accordingly, the Office has failed to establish a *prima facie* case of obviousness and, for at least this reason, this claim is allowable.

Claims 21-26 depend from claim 20 and, as such, are allowable as depending from an allowable base claim. These claims are also allowable for their own recited features which, in combination with those recited in claim 20, are neither shown nor suggested by the references of record, either singly or in combination with one another.

Claims 27-34

Claim 27 recites a method of manipulating an XML document comprising [emphasis added]:

- ***defining one or more crystals***, each of which containing one or more behaviors and an XSLT transformation for transforming an XML document into a DHTML view;
- ***using the one or more crystals*** to render a DHTML view from an XML document;
- enabling user interaction with the DHTML view; and
- mapping, via the one or more behaviors, user interactions in the DHTML view to the XML document.

In making out the rejection of this claim, the Office admits that Kutay does not explicitly teach one or more crystals, each of which containing one or more behaviors and an XSLT transformation. Applicant agrees. The Office then argues that Derose teaches "XPath is the result of an effort to provide a common syntax and semantics for functionality shared between XSL Transformations [XSLT] and XPointer." The Office cites to page 1, section 1, Introduction, to support its argument, the entire section of which is reproduced below:

> 1 Introduction
>
> XPath is the result of an effort to provide a common syntax and semantics for functionality shared between XSL Transformations [XSLT] and XPointer [XPointer]. The primary purpose of XPath is to address parts of an XML [XML] document. In support of this primary purpose, it also provides basic facilities for manipulation of strings, numbers and booleans. XPath uses a compact, non-XML syntax to facilitate use of XPath within URIs and XML attribute values. XPath operates on the abstract, logical structure of an XML document, rather than its surface syntax. XPath gets its name from its use of a path notation as in URLs for navigating through the hi rarchical structure of an XML document.

In addition to its use for addressing, XPath is also designed so that it has a natural subset that can be used for matching (testing whether or not a node matches a pattern); this us of XPath is described in XSLT.

XPath models an XML document as a tree of nodes. There are different types of nodes, including element nodes, attribute nodes and text nodes. XPath defines a way to compute a string-value for each type of node. Some types of nodes also have names. XPath fully supports XML Namespaces [XML Names]. Thus, the name of a node is modeled as a pair consisting of a local part and a possibly null namespace URI; this is called an expanded-name. The data model is described in detail in [5 Data Model].

The primary syntactic construct in XPath is the expression. An expression matches the production Expr. An expression is evaluated to yield an object, which has one of the following four basic types:

- node-set (an unordered collection of nodes without duplicates)
- boolean (true or false)
- number (a floating-point number)
- string (a sequence of UCS characters)

Expression evaluation occurs with respect to a context. XSLT and XPointer specify how the context is determined for XPath expressions used in XSLT and XPointer respectively. The context consists of:

- a node (the context node)
- a pair of non-zero positive integers (the context position and the context size)
- a set of variable bindings
- a function library
- the set of namespace declarations in scope for the expression

The context position is always less than or equal to the context size.

Th variabl bindings consist of a mapping from variable names to variable values. The valu of a variabl is an object, which can b of any of the types that are possible for

the value of an expr ssion, and may also be of additional types not specified here.

The function library consists of a mapping from function names to functions. Each function takes zero or more arguments and returns a single result. This document defines a core function library that all XPath implementations must support (see [4 Core Function Library]). For a function in the core function library, arguments and result are of the four basic types. Both XSLT and XPointer extend XPath by defining additional functions; some of these functions operate on the four basic types; others operate on additional data types defined by XSLT and XPointer.

The namespace declarations consist of a mapping from prefixes to namespace URIs.

The variable bindings, function library and namespace declarations used to evaluate a subexpression are always the same as those used to evaluate the containing expression. The context node, context position, and context size used to evaluate a subexpression are sometimes different from those used to evaluate the containing expression. Several kinds of expressions change the context node; only predicates change the context position and context size (see [2.4 Predicates]). When the evaluation of a kind of expression is described, it will always be explicitly stated if the context node, context position, and context size change for the evaluation of subexpressions; if nothing is said about the context node, context position, and context size, they remain unchanged for the evaluation of subexpressions of that kind of expression.

XPath expressions often occur in XML attributes. The grammar specified in this section applies to the attribute value after XML 1.0 normalization. So, for example, if the grammar uses the character <, this must not appear in the XML source as < but must be quoted according to XML 1.0 rules by, for example, entering it as `<`. Within expressions, literal strings are delimited by single or double quotation marks, which are also used to delimit XML attributes. To avoid a quotation mark in an expression being interpreted by the XML processor as terminating the attribute value the quotation mark can be entered as a character reference (`&quot;` or `&apos;`). Alternatively, the expression can use single quotation marks if the XML attribute is delimited with double quotation marks or vice-versa.

One important kind of xpression is a location path. A location path sel cts a set of nodes relative to the context node. The result of evaluating an expression that is a

> location path is the node-set containing the nodes s l cted by the location path. Location paths can r cursively contain expressions that ar us d to filter sets of nodes. A location path matches the production LocationPath.
>
> In the following grammar, the non-terminals QName and NCName are defined in [XML Names], and S is defined in [XML]. The grammar uses the same EBNF notation as [XML] (except that grammar symbols always have initial capital letters).
>
> Expressions are parsed by first dividing the character string to be parsed into tokens and then parsing the resulting sequence of tokens. Whitespace can be freely used between tokens. The tokenization process is described in [3.7 Lexical Structure].

Applicant respectfully submits that Derose does not teach or suggest ***one or more crystals*** as Applicant has defined and used the term. For a clarification of this terminology, the Office is respectfully referred to Applicant's specification at page 19, line 2, through page 21, line 2, reproduced below, for a discussion of Applicant's inventive concept of crystals:

> In one described embodiment, the notion of a crystal is introduced to enable interactions with a DHTML view to be directly mapped back to the XML file or tree. Advantageously, the crystals are configured to work on various data shapes, independent of the XML schemas. This means that when the data has a particular shape, as defined by the XML tree that contains the data, specific crystals that are configured for that particular shape can be used to render the DHTML and also ensure that user interactions with the DHTML view are directly mapped back to the XML tree. The crystals do not care about the specific data that is provided by the data shape, nor the schema or tags that are used to contain the data.
>
> Consider, for example, Fig. 4 which shows an XML document 400, a crystal 402 and the resultant DHTML document 404. In one basic form, a crystal comprises one or more behaviors 406 and the basic XSL-T 408 that is utilized to transform the XML into the DHTML. The behaviors are implemented, in this particular example, as binary code that is associated with or attached to the DHTML tags that are generated by the XSL-T. Consider, for example, the hierarchical tree that is shown directly below XML document 400. This hierarchical tree represents a portion of an XML tree that is maintained in memory. In this example, the tree has a "products" root node and a "product 1" node that is a child of the

"products" root node. Underneath the "product 1" node are three children nodes labeled "name", "quantity", and "price". This XML tree may thus represent a portion of a purchase order that is utilized to purchase various products. When rendered by the crystal 402, the resulting DHTML view is shown at 410. This DHTML view is diagrammed directly above view 410 as a tree with a behavior associated with a DHTML tag. The DHTML view is essentially a table that contains data that is provided by the XML document. Assume now that a user wishes to modify the purchase order by adding an additional product with a corresponding quantity and price. In the past, the solution to this problem might be to hardcode a function that added a specific "product tag" to the XML and then, correspondingly, to the DHTML view. This is a very inflexible solution that is tied specifically to the schema and tags of the XML document. In the described example, modification of the XML document takes place via the behavior or behaviors that are associated with the crystal 402. Specifically, the behavior that is defined for this particular XML tree structure includes the modifications that can be made to the XML document and a mapping that maps the changes to the DHTML view using application of XSL-T. This behavior is data shape-dependent and not schema- or data-dependent.

This is diagrammatically illustrated in Fig. 4 by the DHTML tree structure shown underneath the DHTML view 404. There, a node corresponding to the "product" node is shown adorned with a behavior. This behavior is binary code that enables a user to interact, via an appropriate UI (such as an in document "add product" button 411 attached to the table) with the DHTML view and have any defined modifications made by the user mapped back to the appropriate XML tree. When a user interacts with the DHTML view, the XML tree is structurally manipulated (as by adding the appropriate tags and structure), and then the XSL-T is invoked to redisplay the DHTML view.

Because neither Kutay nor Derose disclose or suggest ***defining one or more crystals***, each of which containing one or more behaviors and an XSLT transformation for transforming an XML document into a DHTML view, neither reference can possibly disclose or suggest ***using the one or more crystals*** to render a DHTML view from an XML document. Accordingly, the Office has failed to establish a *prima facie* case of obviousness and, for at least this reasons, this claim is allowable.

Claims 28-34 depend from claim 27 and, as such, are allowable as depending from an allowable base claim. These claims are also allowable for their own recited features which, in combination with those recited in claim 27, are neither shown nor suggested by the references of record either singly or in combination with one another. In addition, given the Office's failure to establish a *prima facie* case of obviousness, the rejection of claims 30 and 31 over Clark is not seen to add anything of significance.

Claims 35-38

Claim 35 recites one or more computer-readable media having computer-readable instructions thereon which, when executed by a computer, cause the computer to [emphasis added]:

- ***provide multiple crystals***, each of which containing one or more behaviors and an XSLT transformation for transforming an XML document into a DHTML view;
- ***use one or more of the crystals*** to render a DHTML view from an XML document;
- attach at least one behavior to at least one DHTML tag;
- ascertain that a user has interacted with a DHTML view associated with the at least one DHTML tag; and
- use the behavior associated with the at least one DHTML tag to map a user interaction back to the XML document and make associated structural changes in the XML document.

In making out the rejection of this claim, the Office argues that "claim 35 is directed to a computer-readable media for performing the method of claim 27 and 39 and are [sic] similarly rejected under the same rationale." Applicant respectfully points out that claim 35 is an independent claim that merits an independent examination. Nevertheless, in the interest of furthering prosecution, Applicant will attempt to respond to the Office's rejection of this claim.

In making out the rejection of claim 27, the Office admits that Kutay does not explicitly teach one or more crystals, each of which containing one or more behaviors and an XSLT transformation. Applicant agrees. The Office then argues that Derose teaches "XPath is the result of an effort to provide a common syntax and semantics for functionality shared between XSL Transformations [XSLT] and XPointer." The Office cites to page 1, section 1, Introduction, in support of its argument, the entire section of which is reproduced above.

Applicant respectfully submits that Derose does not teach or suggest one or more computer-readable media having computer-readable instructions thereon which, when executed by a computer, cause the computer to ***provide multiple crystals***, as Applicant has defined and used the term "crystal." The Office is respectfully referred to Applicant's specification at page 19, line 2, through page 21, line 2, reproduced above, for a discussion of Applicant's inventive concept of crystals.

Because neither Kutay nor Derose disclose or suggest the subject matter discussed above, neither reference can possibly disclose or suggest ***using the one or more crystals*** to render a DHTML view from an XML document. Accordingly, the Office has failed to establish a *prima facie* case of obviousness and, for at least these reasons, this claim is allowable.

Claims 36-38 depend from claim 35 and, as such, are allowable as depending from an allowable base claim. These claims are also allowable for their own recited features which, in combination with those recited in claim 35, are neither shown nor suggested by the references of record, either singly or in combination with one another.

Claims 39-45

Claim 39 recites a method of manipulating an XML document comprising [emphasis added]:

- associating one or more behaviors with a DHTML tag in a DHTML view that has been rendered from an XML document; and
- ***responsive to a user interacting with a DHTML view*** associated with the DHTML tag, using the one or more behaviors to map user interactions to the XML document and ***effect structural changes on the XML document.***

In making out the rejection of this claim, the Office argues that Kutay teaches "HTML page, which supports event-based input mechanisms and contains special tags interpretable by the server 210. Alternatively, views 432 may be presented in extensible Markup Language (XML). In one embodiment, each XML view 432 is an XML document accessible to users . . . includes a mechanism for triggering an action 434 and sets of data transmitted from the data model structures 425 and formatted for the type of view, for example in JSP or XML formats. In one embodiment, actions 434 reside within presentation layer 430 and provide a linkage between users 205 and processes 428. Each action 434 is coupled to one or more views 432 that can trigger that action. Also, each action 434 is further coupled to a process 428 triggered by the action and to a set of views 434 that must be activated after the process 428 concludes." The Office cites to paragraphs 58 and 59 in support of its argument, both of which are reproduced below [emphasis added]:

> [0058] Referring back to FIG. 4A, in one embodiment, presentation layer 430 includes multiple views 432 which allow users 205 to ***view processed data***. In one embodiment, views 432 are Java Server Page (USP) views. Each JSP view 432 is a dynamic page, for example an HTML page, which supports event-based input mechanisms and contains special tags interpretable by the server 210. Alternatively, views 432 may be presented in eXtensible Markup Language (XML). In

> one embodiment, each XML view 432 is an XML document accessible to users 205 via Universal Resource Locators (URLs).
>
> [0059] Each view 432 includes a mechanism for triggering an action 434 and sets of data transmitted from the data model structures 425 and formatted for the type of view, for example in JSP or XML formats. In one embodiment, actions 434 reside within presentation layer 430 and provide a linkage between users 205 and processes 428. Each action 434 is coupled to one or more views 432 that can trigger that action. Also, each action 434 is further coupled to a process 428 triggered by the action and to a set of views 434 that must be activated after the process 428 concludes.

The Office asks Applicant to compare the cited excerpts with Applicant's claimed features, namely, "associating one or more behaviors with a DHTML tag in a DHTML view that has been rendered from an XML document; and responsive to a user interacting with a DHTML view associated with the DHTML tag, using the one or more behaviors to map user interactions to the XML document and effect structural changes on the XML document."

Applicant has done as the Office requests and respectfully submits that Kutay does not teach or suggest using one or more behaviors to map user interactions to the XML document and ***effect structural changes on the XML document responsive to a user interacting with a DHTML view*** associated with a DHTML tag. Rather, Kutay discloses multiple views which allow users to ***view processed data***. Accordingly, for at least this reason, this claim is allowable.

Claims 40-45 depend from claim 39 and, as such, are allowable as depending from an allowable base claim. These claims are also allowable for their own recited features which, in combination with those recited in claim 39, are neither shown nor suggested by the references of record, either singly or in combination with one another. In addition, given the allowability of the base claim, the rejection of claims 40, 41, and 43 over the combination with Derose,

and of claims 42, 44, and 45 over the combination with Derose and Clark, is not seen to add anything of significance.

Conclusion

All of the claims are in condition for allowance. Accordingly, Applicant requests a Notice of Allowability be issued forthwith. If the Office's next anticipated action is to be anything other than issuance of a Notice of Allowability, Applicant respectfully requests a telephone call for the purpose of scheduling an interview.

Respectfully submitted,

Dated: 5/18/04

By: RR Cutter, reg #52,772 for Lance R. Sadler
Lance R. Sadler
Reg. No. 38,605
(509) 324-9256